A Reconfigurable Array Based Prototype of a Specialised String Lookup Chip

Vukašin Pejović, Rocío Gómez, Slobodan Bojanić and Juan Guillermo Lalinde

Abstract— **Different strategies for performing string lookups have been developed and deployed during the evolutionary scientific process. These are the results of both the development of technology and the need for improvement of previously existing solutions. Hence, the string lookup problem has been well studied and the respectful amount of good solutions is present. Due to nature of the problem, most of the solutions are software based. Nevertheless, in the modern computing environments, in which the amount of data to be searched trough is increasingly growing, the problem re-arises demanding for the different type of approaches that could target multi-gigabit throughput rates so as to perform close to real-time string lookups. In that light, this paper studies the potential of migrating, a well-known and widely used, Boyer-Moore string lookup algorithm to a hardware specific device capable of satisfying the demanded throughput, by proposing and characterising the initial implementation option on a reconfigurable platform.**

I. Introduction

String Lookup problem is tightly coupled with the notion of a string by itself. It can be simplified as: the fastest way of establishing the number and positions of all the occurrences of a certain sting in another string. The string sought for is usually shorter in length and is named "a pattern". The other string usually significantly longer is called "a text". These terms shall be used throughout this paper.

An ordinary user of a PC, even on a daily bases, runs a number of string lookups, either searching for a name of a file or some content within the file. Therefore the string lookup is omnipresent. In the daily usage the user is generally ready to tolerate certain amount of latency before it is presented with a result. But, if we were to take a closer look at antivirus applications execution, ongoing with other processing power demanding operations, we shall undoubtedly note that antiviruses run for over an hour. With the growing size of antivirus data bases and the growing amount of data stored on a hard disk the need for speeding up the string lookup emerges naturally. If to this example, we add the fact that high speed networking intrusion detection systems, Snort [9] as an example, are also based on string lookups and cannot keep up with the throughput increases introduced by the IEEE 802.3an [12] standard over copper, bringing 10Gbps Ethernet, an additional motive for the closer study of the improvements of the existing string lookup algorithms is identified.

Full set of algorithms was designed to perform the lookup. Starting from naïve brute-force approach in which all the letters of both pattern and text are compared, over more sophisticated algorithms that use different heuristics to speed up the search, such Knuth-Morris-Pratt (KMP) [2] and Boyer-Moore (BM) [1]. The last one is probably the most well known as it is the one most deployed. It is of special interest in high throughput applications since it is used as a part of Snort [9] real time Intrusion Detection system. Different algorithms combining both BM and KMP strategies were proposed since, Horspool [3] being probably the best known. All the mentioned algorithms and their respective implementations, customisations and modifications are run in software as services of operative system or in similar context. Thus, the execution can only be sequential and remains constrained by other aspects of software execution related to scheduling etc. An approach to use different forms of hardware accelerators to bring the speed-up, hence, makes perfect sense.

The related work, that deals with the migration of the string-lookup algorithms to hardware was mostly aiming at the improvement of the intrusion detection systems [6], wherein the string lookup is the fundamental part and the bottle-neck of these systems. In that context, the attempts of naïve string lookup and KMP string lookup in hardware have been studied and documented. The brute force approach is studied in the work of Sourdis and Pnevamtikatos [4], while the KMP work is present in the work of Baker and Prasanna [5]. In the light of the cited papers, this paper studies an initial step forward towards the migration, and thus utilisation, of the Boyer-Moore algorithm in the hardware domain, by presenting and FPGA targeted implementation as the first prototype.

This paper continues by giving the short reminder of the basic BM algorithm, then by explaining the modifications that lead to the hardware implementation of the same algorithm in Section II. The same section also compares the two approaches by presenting the execution time measurements of the software and hardware prototyped versions of the algorithm. The execution

V. Pejović, R. Gómez and S. Bojanić are with the Department of Electronics, of Escuela Tcnica Superior de Inganieros de Telecomunicain of Universidad Politcnica de Madrid, Ciudad Universitaria s/n, 28040 Madrid, Spain. E-mail: vule@die.upm.es

J. G. Lalinde is with the Department of Informatics and Sistems, Escuela de Ingeniería, Universidad EAFIT, Cra. 49 N 7 Sur 50, Medellín, Colombia. E-mail: jlalinde@eafit.edu.co

978-1-4244-1881-7/08/$25.00 © 2008 IEEE

times were measured on a representative test Text and Pattern samples. Views at the future works together with the conclusions in Section 3 close the paper.

II. Boyer Moore based approach

A. Original Algorithm

This subsection explains, in short, the original BM algorithm, by giving an example.

Thence, assuming that the Pattern has a value *conf* and the Text has a value *reconfigurable*, the Pattern length, marked with M has the value of *4*, whilst the length of the Text, marked with N has a value of *14*. Letter case is of no importance for the purposes of this example. BM starts the search by aligning the beginnings of both elements and first comparing the last character of the Pattern to the aligned character of the Text. If a match ocurrs in this comparison the algorithm compares the one character before the last character of the Pattern with the one that aligns with it in the Text, and continues to do so while all the consecutive characters are matched and until the full instance of Pattern in Text has been found. Else, in the case of a mismatch in the first comparison, the algoritm will compare all the remaining characters of the Pattern to the Text character from the first comparison. If none of the characters from the Pattern are mathced the algorithm moves M positions. If, on the contrary, a match is encountered the algoritm would jump to that position, which is by its nature less then M.

In the case of the example values this looks like following and is illustrated in Fig. 1:

- First two characters compared are *o* from Text and *f* from Pattern. No match is found, hence all the remaining Pattern characters are compared to *o*.
- Since there is an *o* in Pattern, the two will be a match, and then the algoritmh determines a jump so as to align two matched characters. This jump has a value of *2* characters.
- After this jump a full match is found, by matching the Pattern in character by character fashion from its end. And the next jump has a value of four since the full match was found.
- Two next steps give no matches at all, and the algorithm jump by *4*, which is actually M until processing the entire Text.

Having explained the jumping actions BM performs, it is obvious that the best case time complexity is *O(N/M)*. Worst case time complexity has been shown not to be greater than *O(3N)* [10]. Table I summarises the results.

B. Hardware Implementation

A next step towards a string look-up chip prototype was the porting of the BM algorithm to a hardware platform. With this goal in mind, a Field Programmable

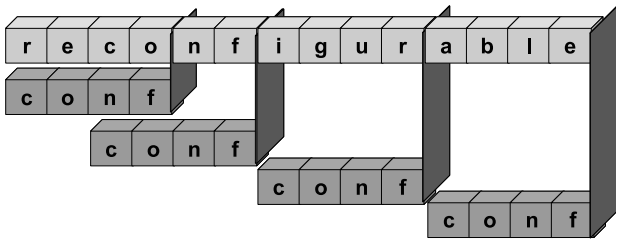

Fig. 1. Original BM algorithm

Gate Array (FPGA) chip, XC4VFX12 [7] from Xilinx was used for rapid prototyping of the device. This chip is present on the Xilinx ML403 [8] development board, which was accessible in our laboratory. This, hence permitted to test the FPGA prototype in practice. The design path used for the implementation consisted in writing the VHDL code, then synthesizing it in Synplify Premier 8.9, and then running place and route and bit file generation phases in Xilinx ISE foundation 9.2. In this way, during the whole implementation phase it was possible to download the design directly to the board and to physically verify the desired functionalities.

Upon the establishment of the destination platform the main actions of the BM algorithm have been ported to VHDL resulting in the string look-up functionality. The illustration of the realised implementation structure is in Fig. 2. Besides the Pattern and the Text blocks, which can be seen as input values of the implementation, and present storage structures, namely registers, the implementation has three more functional blocks: Comparison, Coder and Shift-in Control.

Comparison block performs the character matching functions. This block benefits greatly from the fact that hardware by its nature allows for the usage of multiple concurrent actions. Simply, the last fact is exploited to perform a number of comparison simultaneously and not in cycle-by-cycle fashion. Obvious consequence of this is the increase of the area consumption. Hence, the Comparison block contains the area for speed-up tradeoff, and directly influences on the time complexity of the hardware based BM implementation.

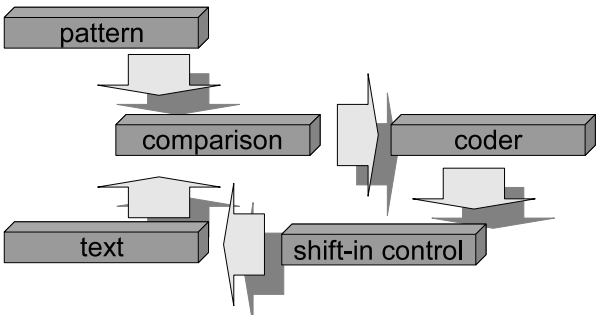

Fig. 2. Structure of the hardware implemented BM

Coder block interprets the results supplied by the Comparison block. Depending on the matches found by the Comparison block, Coder block is able to calculate the value of the jump. The jump essentially represents the amount of data that is to be read from the Text, so that the comparisons could be performed in the next cycle.

Shift-in control block has a role of ensuring that the amount of data asked by Coder is fully provided. In essence, it is a down-counter. This control is necessary, because it is a common practice to supply data form on-chip FIFO. In the case of Text, which can be of almost unlimited length, this is especially the case, hence this was embedded as the basic part of implementation.

Comparison block using concurrent capabilities, executes two steps of comparisons. First, it verifies if the last character of Pattern is found on the correct position in Text. If so, it concurrently checks if the whole Pattern is also found. Else it checks if the Text character that did not match the last character of Pattern matches any other character from the Pattern. These two comparison phases are pipe-lined, hence a single clock cycle delay is induced.

Structurally, the implementation mimics the original BM algorithm. Yet, the way the comparisons are performed has a direct influence at the time complexity of the implementation. The best case time complexity, similarly as in the original case is *O(N/M+1)*. Additional 1 is due to the latency induced in Comparison block. Worst case complexity, however, has a value of *O(N+1)*. Additional 1 is, again, due to latency in Comparison block, and the basic value *N* is owned to the fact that the hardware implementation is spending no more than one clock processing each character of Text. Table I summarises the results.

TABLE I
TIME COMPEXITIES OF SOFTWARE AND HARDWARE IMPLEMENTATIONS OF BM ALGORITHM

Conditions	Software	Hardware
best case	*O(N/M)*	*O(N/M + 1)*
worst case	*O(3N)*	*O(N+1)*

C. Performance comparison

With an aim to characterise the gains that are expected to be induced according to the complexity calculations, the performance was tested in practice. The setup included: the mentioned ML403 board with Virtex4FX12 chip, programmed to behave as described in previous subsection and a PC with AMD Athlon 64 microprocessor at 2Ghz and with 1GB of RAM memory, running BM C code written according to the original algorithm, under Windows XP operative system. Both the platform were expected to perform the same string look-up task, whilst the execution time needed to obtain the results was measured.

The look-up task consisted in searching for different Patterns, Table II and Table III, in Text in English, Romeo and Juliet from W. Shakespeare, and in Text in Spanish, La Vida es Sueño, from Pedro Calderón de la Barca. Both text have a lots of repetitive contents and can be a good test-bench for the performance estimation. The first text contains 6204 words, while the second contains 8268 words. Number of characters in the first text is 32681 and in the second 65387. Since different languages have different character usage and probability it was interesting to establish if that would change any parameters in the performance.

TABLE II
EXECUTION TIMES

Pattern	Occurrences	PC	FPGA
Romeo	62	4.475470	0.45553
Juliet	25	4.383440	0.43973
Capulet	46	3.778130	0.42879
Montague	28	3.331720	0.42921
therefore	3	3.058910	0.42303
Enter Rome	2	2.482030	0.41761

Before performing the tests it was necessary to download Texts to the FPGA. This was done by downloading Texts to the internal FPGA memory blocks. FPGA was set to run with the clock of 100MHz, since this frequency was available on the board. In the described surrounding the hardware implemented algorithm outperformed software between 11 and 9 times.

TABLE III
EXECUTION TIMES

Pattern	Occurrences	PC	FPGA
Segis	66	5.309690	0.89899
Clarin	43	4.304060	0.86117
Rosaura	46	4.044840	0.85417
Clotaldo	60	3.08984	0.83287
Segismund	66	3.122500	0.80373
Segismundo	66	3.31750	0.80813

In the last two lines of Table III one could notice how execution times changes depending on the last character of Pattern. Simply, since letter *e* is more probable in Spanish than letter *d*, the addition of the former letter prolongs the execution time.

In addition to the software vs. hardware comparison, the hardware based algorithm was characterised in terms of maximum frequency. The result are given

in Figure 3. Same figure shows how the maximum frequency scales with the length of Pattern. Two cases were considered. First one, plotted with the dotted line, shows the results of the implementation variant with the Text in internal FPGA memory, while the continuous line shows the results when the Text is fed from the outside over FPGA pins.

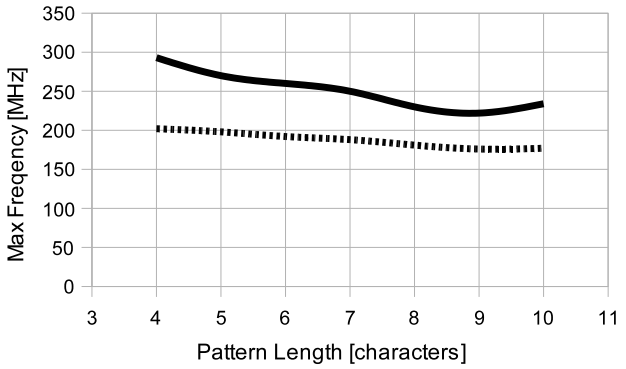

Fig. 3. Performance Characterisation

Since, Shift-in Control block was implemented to read a single character per clock cycle, if the character encoding is ASCII, the throughput of the system can be calculated by multiplying the number of bits in character with maximum frequency. When data is sourced from the outside of the chip, the throughput would scale between 1.8 an 2.4 Gbps. The achievable throughput is not too high, yet the results are produced in initial phase of the work, and there remain a lot of possibilities for improvement.

III. Conclusions and Future Work

A way to port string look-up functionality to a hardware device was presented in this paper. Full prototype, based on reconfigurable logic, was built and its performance was characterised. It was then compared with the performance of the equivalent software based system. In direct comparison the hardware prototype was showed to perform at least 9 times faster. The performance improvement was due to the exploitation of the intrinsic parallelism of hardware based platforms and execution of multiple comparisons simultaneously.

The work, however is still far from an IC prototype. Before getting there a number of problems needs to be tackled. First in the line is certainly the problem of the bottle-neck of the hardware implementation of the system in the data Shift-in part. Besides that, it is of great importance to improve internal performance of the implementation. This would in practice mean that the implementation need to be optimised in order to achieve higher clock rate capabilities.

Another related question should target the multi-pattern look-up. In other words, would it be possible to have a simultaneous look-up of multiple strings in hardware. One possible direction towards this answer is to study Commentz-Walter [11] algorithm.

Independently of the string-lookup algorithm the future IC needs to use some of the high speed IO buses to be capable of delivering the throughput achieved. One possible option could be the PCI Express bus, since it is a point-to-point type of bus it allows a lots of flexibility in combination with 2Gbps troughput per line of communication.

Besides the fact that the quantity of the remaining problems is respectable, the work presents an initial efforts towards the BM based string look-up in hardware, showing it to perform at least nine times faster than the software implemented version of the algorithm. It also gives a path towards further improvements with the goal of reaching 10Gbps boundaries.

Acknowledgment

This work has been partially financed by the Technical University of Madrid under the project #AL07-PID-047 and by Spanish Agency for International Cooperation (AECI) under the project A/010916/07.

References

[1] R.S. Boyer and J.S. Moore, *A Fast String Searching Algorithm*, Communications of the ACM, Vol 20, No 10, pp. 66-72, Oct. 1977.
[2] D.E. Knuth, J.H. Morris, and V.R Pratt, *Fast Pattern Matching in Strings*, SIAM Journal on Computing, Vol 6, No 2, pp. 323-350, June 1977.
[3] R. Horspool, *Practical Fast Searching in Strings*, Software - Practice and Experience, vol. 10, pp. 501-506, 1980.
[4] I. Sourdis and D. Pnevmatikatos, *Fast, Large-Scale String Match for a 10Gbps FPGA-based Network Intrusion Detection System*, 13th International Conference on Field Programmable Logic and Applications (FPL'03), pp. 880-889, Lisbon, Portugal, September 2003.
[5] Z. Baker and V. K. Prasanna, *Time and area efficient pattern matching on FPGAs*, Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate A (FPGA2004), pp. 223-232, February 2004.
[6] V. Pejović, S. Bojanić, C. Carreras, *Structural Framework for High Speed Intrusion Detection/Prevention Signature Based System*, International Journal of Computer Science and Network Security, vol 6. no. 9b, pp. 175-181, Sept. 2006.
[7] Xilinx Inc., *Xilinx Inc. Virtex 4 user guide v2.3*, August 2007.
[8] Xilinx Inc., *ML401/2/3* , May 2007.
[9] Snort - the de facto standard for intrusion detection/prevention, *www.snort.org*
[10] R. Cole, R. Hariharan, U. Zwick, and M.S. Paterson, *Tighter lower bounds on the exact complexity of string matching*, SIAM Journal on Computing, 24(1), pp. 30-45, 1995.
[11] B. Commentz-Walter *A string matching algorithm fast on the average*, Proc. of 6th International Coll. on Automata, Languages, and Programming, pp. 118-132, 1979.
[12] IEEE 802.3an-2006 Standard.